Java PathFinder

A Model Checker for Java Programs

Recipient of NASA's 2003 TGIR Award for Engineering Innovation

JPF is now available as Open Source at SourceForge.net

Abstract

The majority of work carried out in the formal methods community throughout the last three decades has (for good reasons) been devoted to special languages designed to make it easier to experiment with mechanized formal methods such as theorem provers and model checkers. We believe it is time for the formal methods community to shift some of its attention towards the analysis of programs written in modern programming languages.

In keeping with this philosophy we have developed a verification and testing environment for Java which integrates model checking, program analysis and testing - called Java PathFinder (JPF). JPF consists of a custom-made Java Virtual Machine (JVM) that interprets bytecode combined with a search interface to allow the complete behavior of a Java program to be analyzed (including all interleavings of concurrent programs). JPF is implemented in Java and its architecture is highly modular to support rapid prototyping of new features.

JPF is a so-called explicit-state model checker, since it enumerates all visited states, and therefore suffers from the state-explosion problem inherent in analyzing large programs. It is ideally suited to analyzing programs less than 10kLOC, but has been successfully applied to finding errors in concurrent programs up to 100kLOC.

JPF has the following list of main features:

- Works at the bytecode level, hence all of Java can be model checked as well as errors that can only be found on the bytecode level can be uncovered.
- By default it checks for all runtime errors (uncaught exceptions), assertion violations (supports Java's assert), and deadlocks. Although JPF previously supported LTL checking this feature has been deprecated.
- Contains garbage collection, since a typical Java program will be infinite-state without garbage collection (state size will grow infinitely).
- Use symmetry reductions of the heap during model checking to reduce state-explosion.
- Use dynamic partial-order reductions to reduce the number of interleavings analyzed.
- Supports BFS and DFS search as well as heuristic based search, e.g. A*, Best-First, etc.

- Contains a run-time analysis mode that uses the Eraser algorithm to find race-conditions as well as a Lock Order analysis to check whether locks can be taken in different orders (which often leads to deadlocks). The results of the run-time analysis can be used to guide JPF towards errors.
- Includes the capability to do symbolic execution of Java programs, including symbolic execution of complex data (such as linked lists and trees).
- Supports the calculation of structural coverage (branch and statement coverage) during model checking; coverage can also guide heuristic search.
- Facility to explain the real cause of an error the tool finds.
- Allows the creation of listeners that can subscribe to events during the search - this facility makes JPF easily extensible.
- Supports the creation of dedicated code to be executed in place of regular classes - this allows one to easily handle native calls and improve the efficiency of the analysis.

People

Core Ames Team

- Willem Visser
- Peter Mehlitz
- Corina Pasareanu
- John Penix
- Masoud Mansouri-Samani

Student Interns

- Alex Groce
- Sarfraz Khurshid
- Flavio Lerda
- Oksana Tkachuk
- Radek Pelanek
- Jamie Cobleigh
- Nestor Catano
- Jesus Martinez

Collaborators

- Matt Dwyer
- John Hatcliff
- Scott Stoller
- Gerard Holzmann
- Robby
- Radu Iosif

Related Program Model Checking Tools

Last updated: January 14, 2005 by Raymond De Ocampo